US006487716B1

(12) **United States Patent**
Choi et al.

(10) **Patent No.: US 6,487,716 B1**
(45) **Date of Patent: Nov. 26, 2002**

(54) **METHODS AND APPARATUS FOR OPTIMIZING PROGRAMS IN THE PRESENCE OF EXCEPTIONS**

(75) Inventors: **Jong-Deok Choi**, Mount Kisco, NY (US); **Manish Gupta**, Peekskill, NY (US); **Michael Hind**, Cortlandt Manor, NY (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/415,137**

(22) Filed: **Oct. 8, 1999**

(51) **Int. Cl.**[7] **G06F 9/44**
(52) **U.S. Cl.** **717/159**; 717/147; 717/148; 717/159; 712/244
(58) **Field of Search** 717/4–9, 140–161; 707/206; 712/244

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,778,233	A *	7/1998	Besaw et al.	717/9
5,799,179	A	8/1998	Ebcioglu et al.	
6,031,992	A *	2/2000	Cmelik et al.	717/5
6,081,665	A *	6/2000	Nelsen et al.	717/5
6,113,651	A *	9/2000	Sakai et al.	717/6
6,247,172	B1 *	6/2001	Dunn et al.	717/5
6,314,555	B1 *	11/2001	Ndumu et al.	717/1
6,343,375	B1	1/2002	Gupta et al.	

OTHER PUBLICATIONS

U.S. patent application Ser. No. 09/323,989, Choi et al., filed Jun. 2, 1999.

J.–D. Choi et al., "Efficient and Precise Modeling of Exceptions for the Analysis of Java Programs," ACM SIGPLAN –SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, pp. 1–11, Sep. 6, 1999.

C. Chambers et al., "Dependence Analysis for Java," Proceedings of the 12th International Workshop on Languages and Compilers for Parallel Computing, San Diego, 18 pages, Aug. 1999.

B.C. Le, "An Out–of–Order Execution Technique for Runtime Binary Translators," Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 151–158, Oct. 1998.

D.I. August et al., "Integrated Predicated and Speculative Execution in the IMPACT EPIC Architecture," Proceedings of 25th International Symposium on Computer Architecture, pp. 227–237, Jul. 1998.

K. Ebcioglu, "Some Design Ideas for a VLIW Architecture for Sequential Natured Software," Parallel Processing, M. Cosnard et al. (editors), pp. 3–21, North Holland, 1998.

K. Ebcioglu et al., "DAISY: Dynamic Compilation for 100% Architectural Compatibility," Proceedings of 24th International Symposium on Computer Architecture, pp. 26–37, Denver, CO, Jun. 1997.

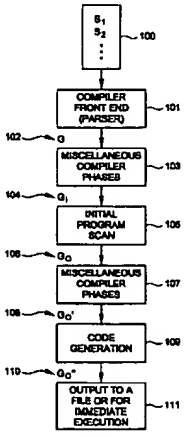
(List continued on next page.)

Primary Examiner—Tuan Q. Dam
Assistant Examiner—Chuck Kendall
(74) *Attorney, Agent, or Firm*—Douglas W. Cameron; Ryan, Mason & Lewis, LLP

(57) **ABSTRACT**

A method and several variants are provided for analyzing and transforming a computer program such that instructions may be reordered even across instructions that may throw an exception, while strictly preserving the precise exception semantics of the original program. The method uses program analysis to identify the subset of program state that needs to be preserved if an exception is thrown. Furthermore, the method performs a program transformation that allows dependence constraints among potentially excepting instructions to be completely ignored while applying program optimizations. This transformation does not require any special hardware support, and requires a compensation code to be executed only if an exception is thrown, i.e., no additional instructions need to be executed if an exception is not thrown. Variants of the method show how one or several of the features of the method may be performed.

25 Claims, 5 Drawing Sheets



OTHER PUBLICATIONS

J. Gosling et al., Chapter 11, Exceptions, "The Java Language Specification (Java Series)," Addison–Wesley Publishing Company, Reading, MA, pp. 201–213, 1996.

A. Aiken et al., "Safe: A Semantic Technique for Transforming Programs in the Presence of Errors," ACM Transactions on Programming Languages and Systems, 17(1):63–84, Jan. 1995.

S. Mahlke et al., "Sentinel Scheduling: A Model for Compiler–Controlled Speculative Execution," ACM Transactions on Computer Systems, 11(4):376–408, Nov. 1993.

K. Ebcioglu et al., "An Architectural Framework for Supporting Heterogeneous Instruction–Set Architectures," IEEE Computer, 26(6), pp. 39–56, Jun. 1993.

P.P. Chang et al., "IMPACT: An Architectural Framework for Multiple–Instruction–Issue Processors," Proceedings of 18th International Symposium on Computer Architecture, pp. 266–275, 1991.

M.D. Smith et al., "Boosting Beyond Static Scheduling in a Superscaler Processor," Proceedings of 19th International Symposium on Computer Architecture, pp. 344–354, May 1990.

* cited by examiner

US-PAT-NO: 6487716

DOCUMENT-IDENTIFIER: US 6487716 B1

TITLE: Methods and apparatus for optimizing programs in the presence of exceptions

DATE-ISSUED: November 26, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Choi; Jong-Deok	Mount Kisco	NY	N/A	N/A
Gupta; Manish	Peekskill	NY	N/A	N/A
Hind; Michael	Cortlandt Manor	NY	N/A	N/A

US-CL-CURRENT: 717/159, 712/244 , 717/147 , 717/148

ABSTRACT:

A method and several variants are provided for analyzing and transforming a computer program such that instructions may be reordered even across instructions that may throw an exception, while strictly preserving the precise exception semantics of the original program. The method uses program analysis to identify the subset of program state that needs to be preserved if an exception is thrown. Furthermore, the method performs a program transformation that allows dependence constraints among potentially excepting instructions to be completely ignored while applying program optimizations. This transformation does not require any special hardware support, and requires a compensation code to be executed only if an exception is thrown, i.e., no additional instructions need to be executed if an exception is not thrown. Variants of the method show how one or several of the features of the method may be performed.

25 Claims, 7 Drawing figures

Exemplary Claim Number: 19

Number of Drawing Sheets: 5

---------- KWIC ---------

Brief Summary Text - BSTX (13):

Many compilers use a representation called a **call graph** to analyze an entire program. A **call graph** has nodes representing procedures, and edges representing procedure calls. We use the term procedure to refer to subroutines, functions, and also methods in object-oriented languages. A **direct** procedure call, where the callee (called procedure) is known at the call site, is represented by a single edge in the **call graph** from the caller to the callee. A procedure **call, where the callee is not known, such as a virtual-method call** in an object-oriented language or an indirect call through a pointer, is represented by edges from the caller to each possible callee. It is also possible that given a particular (callee) procedure, all callers of it may not be known. In that case, the **call graph** would conservatively put edges from all possible callers to that callee.

Brief Summary Text - BSTX (33):

An alternative embodiment of the method uses a different analysis to overcome the write-barrier dependences due to PEIs. In this embodiment, no extra parameter is added to procedures. Instead, the analysis to record the liveness information of variables at enclosing exception is done at compile time, using the **call graph** representation of the program. The information about exception handlers, if any, surrounding a procedure call from A to B is propagated to B and to all procedures reachable in the **call graph** from B (representing all procedures transitively called from B). In this embodiment, there is no need to create a specialized version of the procedure. All write-barrier dependences related to PEIs that do not have a non-trivial enclosing exception handler, are ignored. In the special case where the static analysis shows that a procedure cannot have any dynamically enclosing exception handler with live variables, those regions of code in that procedure which do not have an exception handler block within the procedure, can be viewed as write-barrier-free regions. These regions are optimized while ignoring all write-barrier dependences.

Detailed Description Text - DETX (8):

Lines 201-236 describe the Initial Program Scan phase. Line 202 builds a **call graph** representation of the program. Lines 203-210 determine what predefined exceptions may be thrown **directly** by a procedure and represent this information in the bit vector Except for that procedure. Lines 225-236 later propagate this information interprocedurally using a worklist so that each caller includes the list of exceptions thrown by any callee procedure that is called by it. Lines 211-216 determine the behavior of each procedure that may finally handle an exception not caught by any exception handler. For example, in the Java.TM. language, an exception which is not caught by an exception handler is handled by executing a special procedure, the uncaughtException method of the thread group. Java.TM. also allows user code to override the uncaughtException method. Line 213 checks if the default handler is non-trivial (as defined by Lines 301-306. If so, EEH.sub.init is initialized to 1, which in turn would cause all bits of EEH.sub.c to be initialized to 1 on line 218. Lines 217-224 determine which call sites are enclosed by exception handlers that catch a specific exception and are non-trivial (as defined by lines 301-306). This information is stored in the bit vector for that call site, EEH.sub.c. In a preferred embodiment, this information is propagated across procedures dynamically during execution, as described later.

Detailed Description Text - DETX (26):

Line 701 builds the **call graph** representation of the program. Line 702 performs an interprocedural analysis on this graph to identify the exception handlers that may dynamically enclose each procedure, and is described further below. Line 703 loops over each procedure in the program. Lines 704-708 check if the code region is not possibly enclosed in a non-trivial exception handler for a specific exception type--if so, all write-barrier dependences for that exception type are ignored while performing optimizations.

Detailed Description Text - DETX (34):

An alternative embodiment can save storage by compressing the bit vector to 1 bit, which would represent any predefined exception. This can be computed by performing a bitwise or of the bits in the original bit vector. In another variant, an explicit extra parameter may not be required. Instead the single bit of information is encoded in an unused bit of an existing parameter. For example, the two least significant bits of a pointer field are not required on a machine with 4 byte words. Thus, any existing pointer parameter can be used to encode this information. In particular, **virtual-method calls** always pass a pointer to the object they operate on, called the this pointer, and thus, this parameter can be used to store the 1 bit of information.

Detailed Description Text - DETX (35):

An alternative embodiment modifies the manner in which the dynamic dispatch code is generated during code generation (described currently in line 407). Specifically, when generating the code to determine whether to call normal or specialized code, a decision is possibly made at code-generation time using an enhanced version of the static analysis described in alternative embodiment associated with lines 700-907. In these cases a **direct** call may occur, eliminating the need for dynamic dispatch code (line 407) for that call site. The enhanced static analysis additionally computes for each procedure whether all call sites to it have a call-path where some bit of an EEH is set to 1. This information records what will always occur. There are three cases: A. If the bitwise and of the statically computed any information (computed by the alternative embodiment lines 700-907) for a call site and the called procedures bit vector, Except.sub.m, is all zero then all calls can be made to the specialized version of the procedure. Thus, the need for dynamic dispatch is removed. Instead code is generated to make a **direct** call to the specialized procedure. B. If the bitwise and of the statically computed always information for a call site and the called procedures bit vector, Except.sub.m, is nonzero then all calls can be made to the normal version of the procedure. Thus, once again, the need for dynamic dispatch is removed. Instead code is generated to make a **direct** call to the normal version of the procedure. C. If neither A nor B is applicable, then the dynamic dispatch code described in line 407 is generated. This static analysis occurs after the Initial Program Scan 105, but before the Code Generation phase 109.

Detailed Description Paragraph Table - DETL (1):

201: InitialProgramScan(G) 202: Build **call-graph,** CallG, of program represented by G 203: for each node, p, in CallG do // for each procedure 204: set all bits of Except.sub.p to 0 205: for each instruction, i, in p do 206: for each predefined exception, e, that i may throw do 207: set the e bit of Except.sub.p to 1 208: end do 209: end do 210: end do 211: EEH.sub.init = 0 212: for each default handler h that deals with uncaught exceptions 213: if isNonTrivialHander(h) then 214: EEH.sub.init = 1; break; 215: end if 216: end do 217: for each call site, c, in p do 218: set all bits of EEH.sub.c to EEH.sub.init 219: for each exception handler, h, enclosing c do 220: if isNonTrivialHander(h) then 221: set the h bit for EEH.sub.c to 1 222: end if 223: end do 224: end do 225: Set WorkList = enumeration of nodes in CallG in reverse topological sort order 226: while WorkList not empty do 227: Get first element p and delete it from WorkList 228: for each edge (p, q) in CallG do // for each call to procedure q in p 229: if Except.sub.p != (Except.sub.p.vertline.Except.sub.q) then // ".vertline." denotes bit-wise OR 230: Except.sub.p = Except.sub.p.vertline.Except.sub.q 231: for each edge (r, p) in CallG do // for each caller r of p 232: add r to WorkList if it is not already present 233: end do 234: end if 235: end do 236: end do 301: isNonTrivialHandler(h) 302: if (handler h does not use a variable visible outside the handler other than the exception object and the entry to h is post-dominated by a call to system exit or termination of the current thread) then 303: return false 304: else 305: return true 306: end if

Detailed Description Paragraph Table - DETL (5):

700: CodeGeneration(G) 701: Build **call-graph,** CallG, of program represented by G 702: Perform EnclosingHandlerAnalysis(CallG) 703: for each procedure, m, in program do 704: if(h bit of EnclosingHandlers.sub.m is not set and there is no non- trivial local exception handler for exception type h) then 705: optimize code ignoring write-barrier dependences for each PEI corresponding to h 706: else 707: generate code honoring write-barrier dependences for each PEI corresponding to h 708: end if 709: end do